What Is Object-Orientation?

Based on Chapter 4 of Bennett, McRobb and Farmer:

Object Oriented Systems Analysis and Design Using UML, (3rd Edition), McGraw Hill, 2005.

In This Lecture You Will Learn:

- The fundamental concepts of object-orientation
- The justifications for an object-oriented approach

Basic Concepts

- The main concepts introduced here are:
  - Objects, Classes and Instances
  - Object State
  - Generalization and Specialization
  - Message-passing and Encapsulation
  - Polymorphism

Objects

An object is:

"an abstraction of something in a problem domain, reflecting the capabilities of the system to

- keep information about it,
- interact with it,
- or both."

Coad and Yourdon (1990)

Objects

"Objects have state, behaviour and identity."

Booch (1994)

- *State*: the condition of an object at any moment, affecting how it can behave
- *Behaviour*: what an object can do, how it can respond to events and stimuli
- *Identity*: each object is unique

Examples of Objects

Object	Identity	Behaviour	State
A person.	'Hussain Pervez.'	Speak, walk, read.	Studying, resting, qualified.
A shirt.	My favourite button white denim shirt.	Shrink, stain, rip.	Pressed, dirty, worn.
A sale.	Sale no #0015, 18/05/05.	Earn loyalty points.	Invoiced, cancelled.
A bottle of ketchup.	*This* bottle of ketchup.	Spill in transit.	Unsold, opened, empty.

Class and Instance

- All objects are *instances* of some *class*
- Class:

  a description of a set of objects with similar
  - features (attributes, operations, links);
  - semantics;
  - constraints (e.g. when and whether an object can be instantiated).

  OMG (2004)

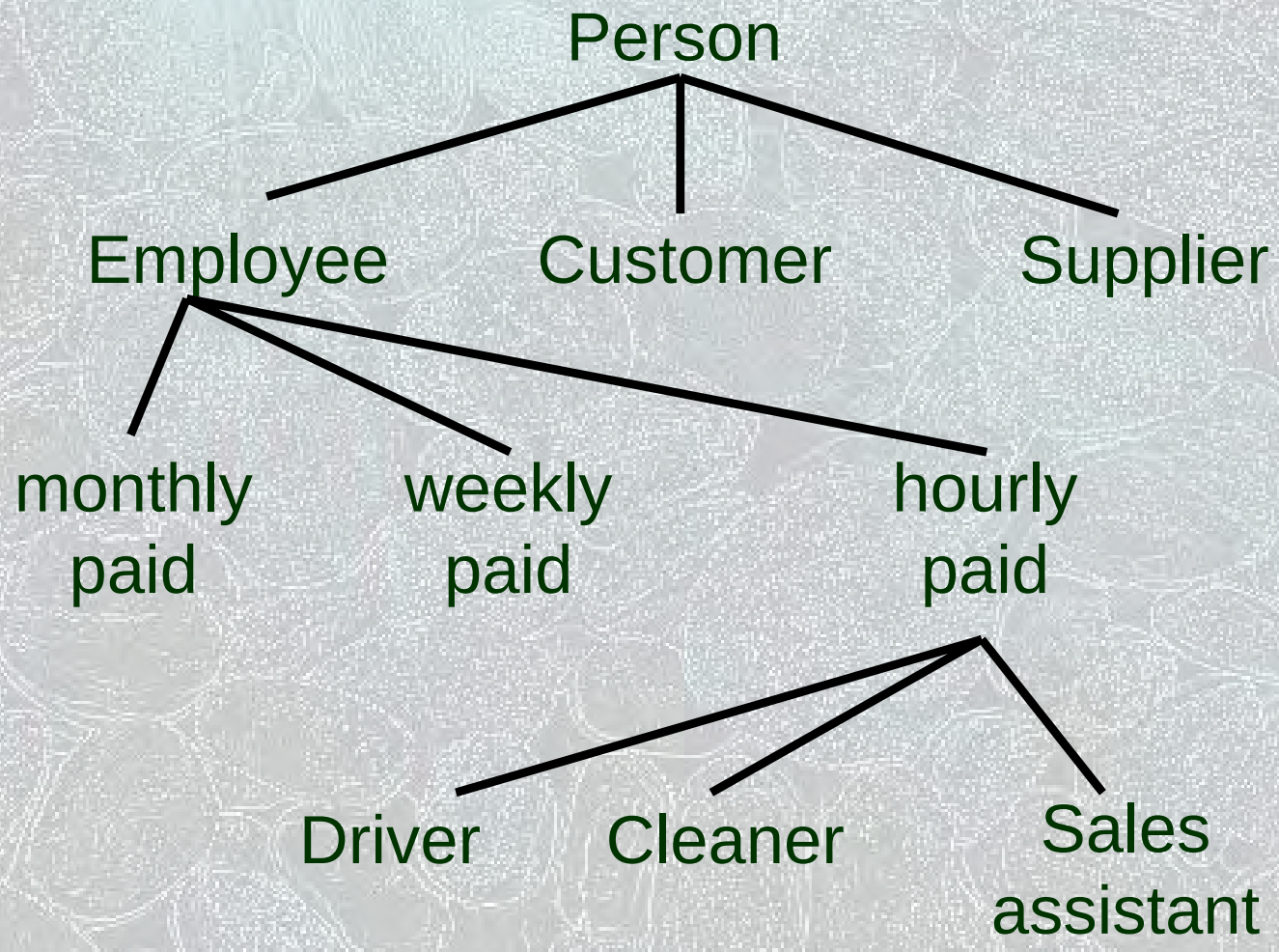
Class and Instance

- An object is an instance of some class
- So, instance = object
  - but also carries connotations of the class to which the object belongs
- Instances of a class are similar in their:
  - *Structure*: what it *knows*, what information it holds, what links it has to other objects
  - *Behaviour*: what an object *can do*

Generalization and Specialization

- Classification is hierarchic in nature
- For example, a person may be an employee, a customer, a supplier of a service
- An employee may be paid monthly, weekly or hourly
- An hourly paid employee may be a driver, a cleaner, a sales assistant

Specialization Hierarchy



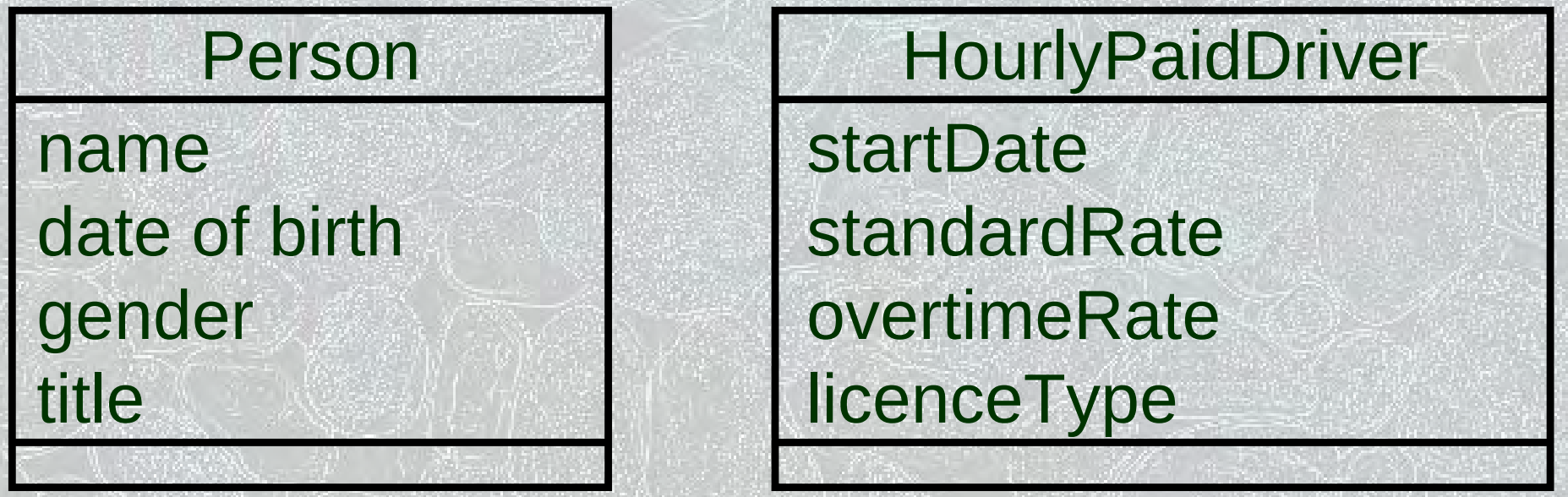
More general (superclasses)

More specialized (subclasses)

Generalization and Specialization

- More general bits of description are *abstracted out* from specialized classes:

General (superclass) **Specialized (subclass)**



Inheritance

- The *whole* description of a superclass applies to *all* its subclasses, including:
  - Information structure (including associations)
  - Behaviour
- Often known loosely as *inheritance*
- (But actually inheritance is how an O-O programming language *implements* generalization / specialization)

Message-passing

- Several objects may collaborate to fulfil each system action
- “Record CD sale” could involve:
  - A CD stock item object
  - A sales transaction object
  - A sales assistant object
- These objects communicate by sending each other messages

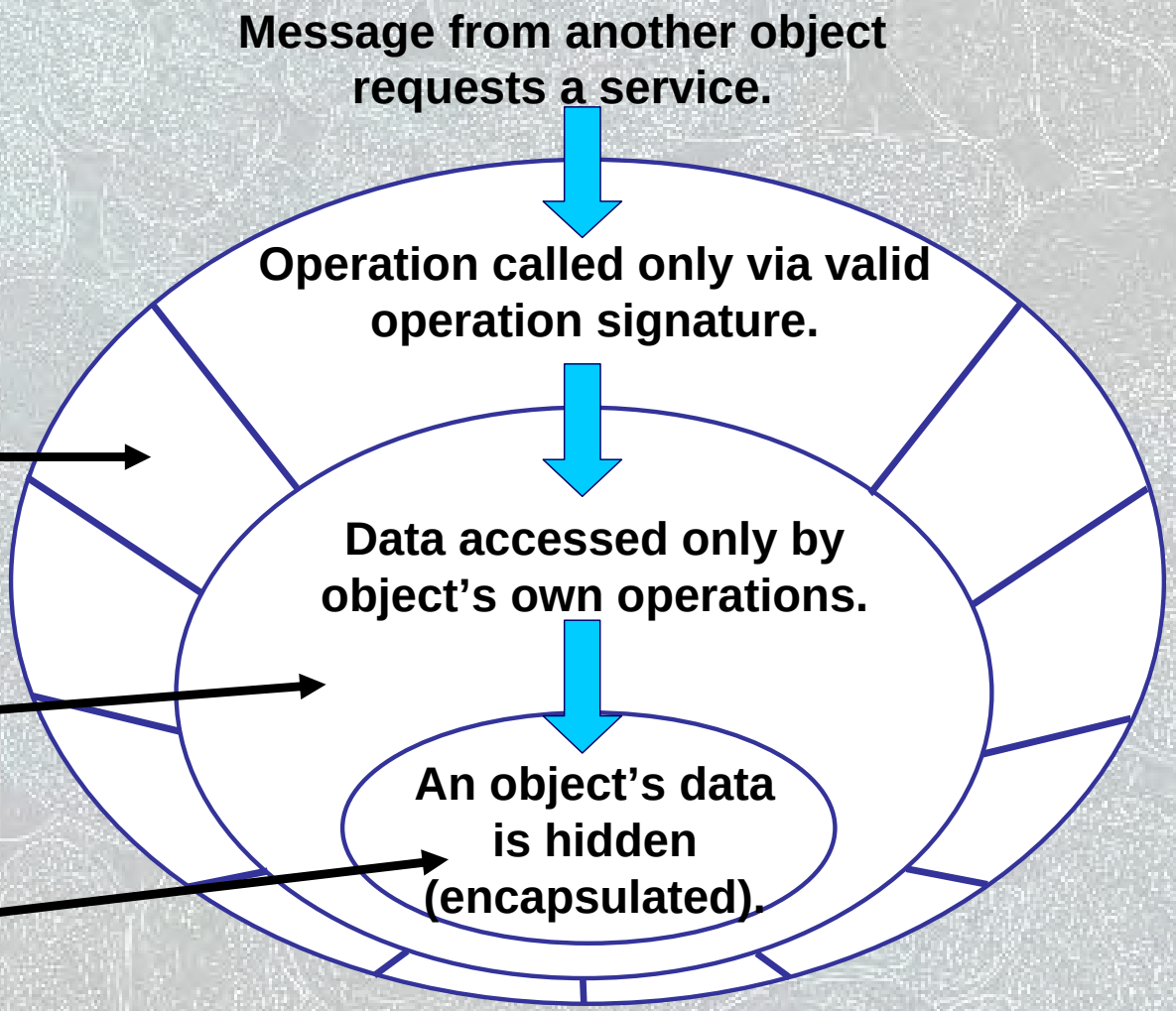
Message-passing and Encapsulation

'Layers of an onion' model of an object:

An outer layer of operation signatures…

…gives access to middle layer of operations…

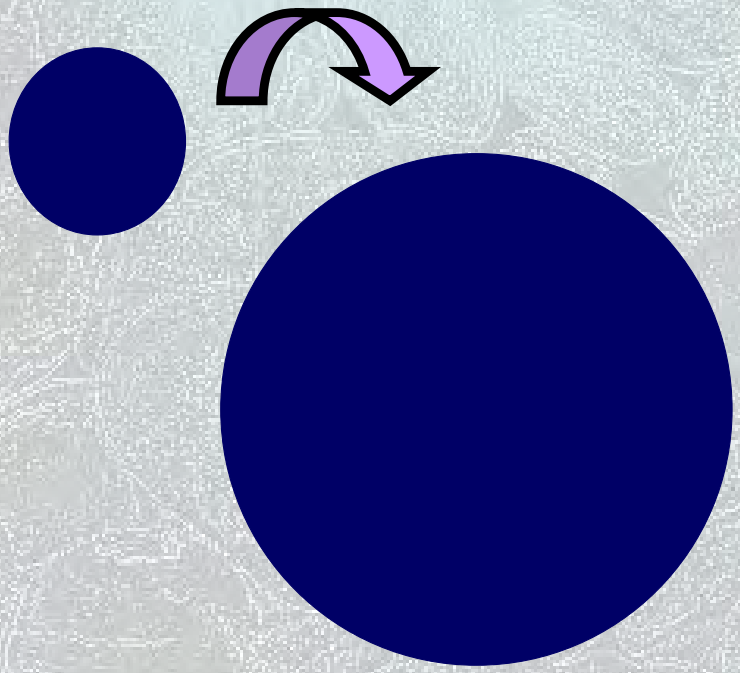
…which can access inner core of data

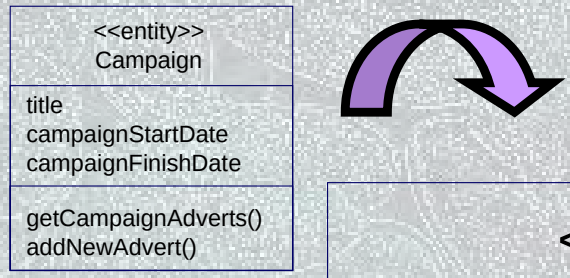


Polymorphism

- Polymorphism allows one message to be sent to objects of different classes
- Sending object need not know what kind of object will receive the message
- Each receiving object knows how to respond appropriately
- For example, a ‘resize’ operation in a graphics package

Polymorphism in Resize Operations





Advantages of O-O

- Can save effort
  - Reuse of generalized components cuts work, cost and time
- Can improve software quality
  - Encapsulation increases modularity
  - Sub-systems less coupled to each other
  - Better translations between analysis and design models and working code

Summary

In this lecture you have learned about:

- The fundamental concepts of O-O
  - Object, class, instance
  - Generalization and specialization
  - Message-passing and polymorphism
- Some of the advantages and justifications of O-O

References

- Coad and Yourdon (1990)
- Booch (1994)
- OMG (2004)

(For full bibliographic details, see Bennett, McRobb and Farmer)

Modelling Concepts

Based on Chapter 5 of Bennett, McRobb and Farmer:

Object Oriented Systems Analysis and Design Using UML, (3rd Edition), McGraw Hill, 2005.

In This Lecture You Will Learn:

- What is meant by a model
- The distinction between a model and a diagram
- The UML concept of a model

What is a Model

- Like a map, a model represents something else
- A useful model has the right level of detail and represents only what is important for the task in hand
- Many things can be modelled: bridges, traffic flow, buildings, economic policy

Why Use a Model?

- A model is quicker and easier to build
- A model can be used in a simulation
- A model can evolve as we learn
- We can choose which details to include in a model
- A model can represent real or imaginary things from any domain

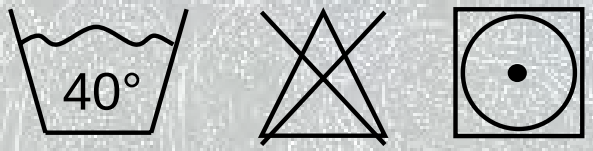
Modelling Organizations

Organizations are human activity systems.

- The situation is complex
- Stakeholders have different views
- We have to model requirements accurately, completely and unambiguously
- The model must not prejudge the solution

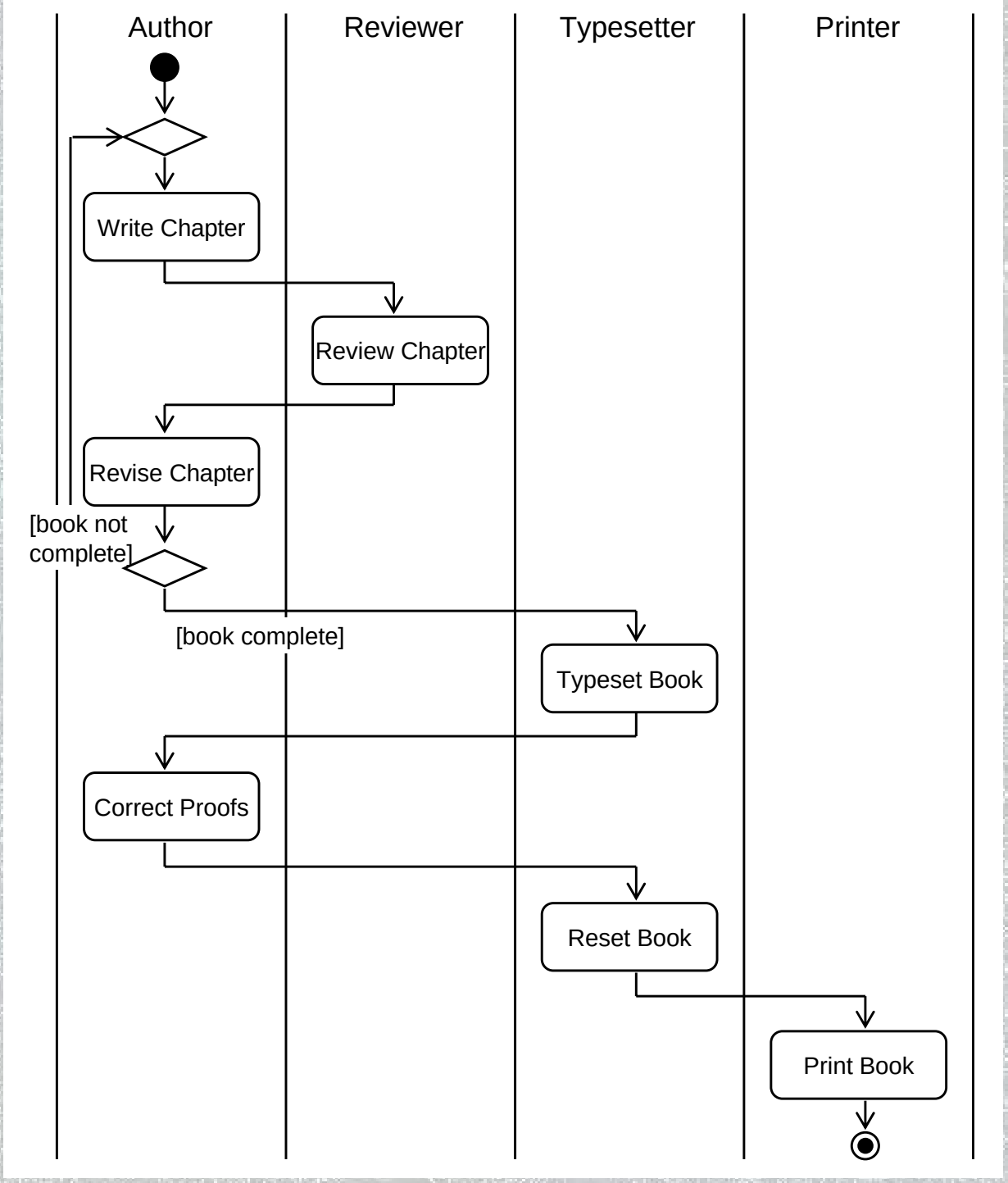
What is a Diagram?

- Abstract shapes are used to represent things or actions from the real world
- Diagrams follow rules or standards
- The standards make sure that different people will interpret the diagram in the same way

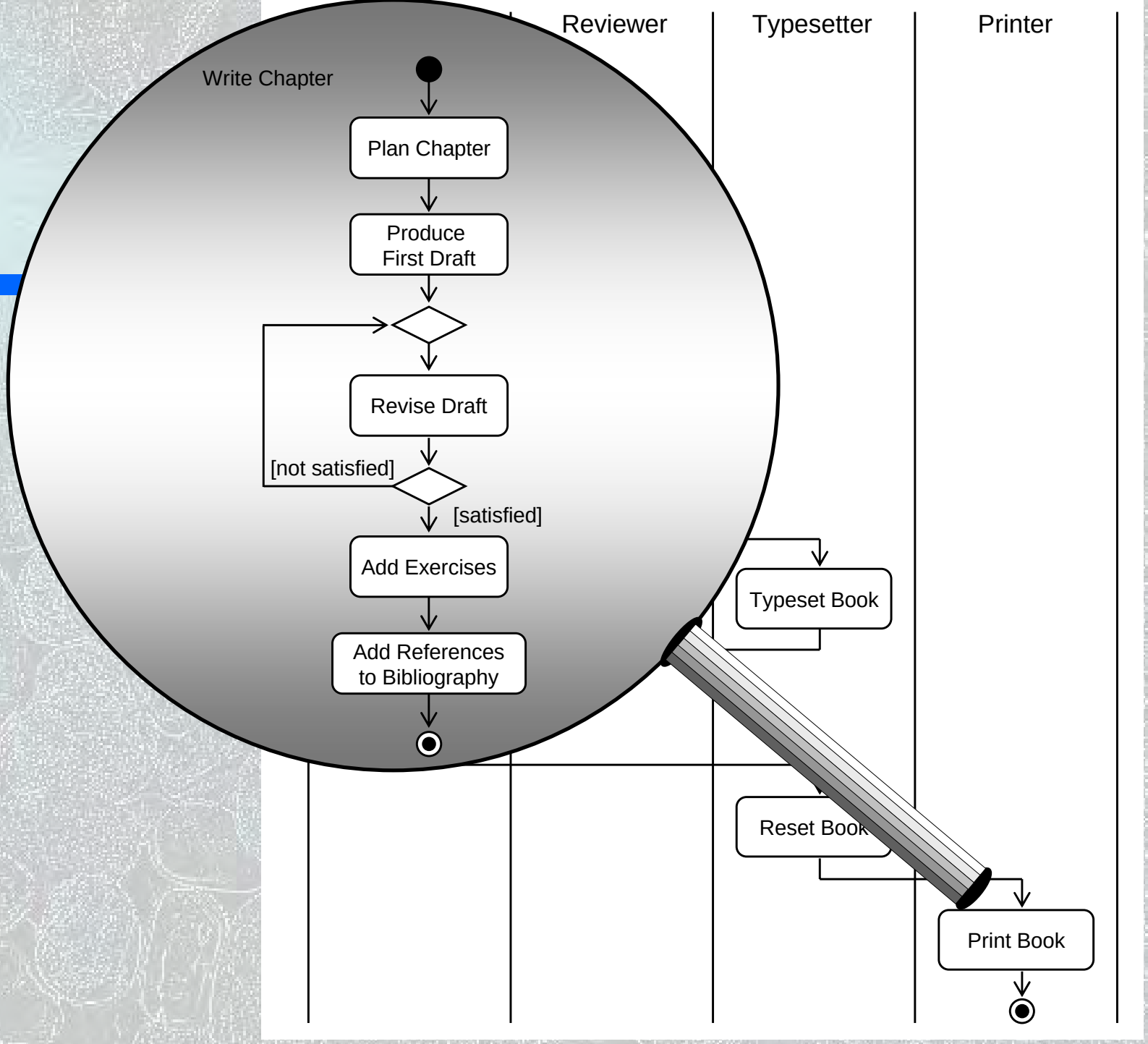


An Example of a Diagram

- An activity diagram of the tasks involved in producing a book.

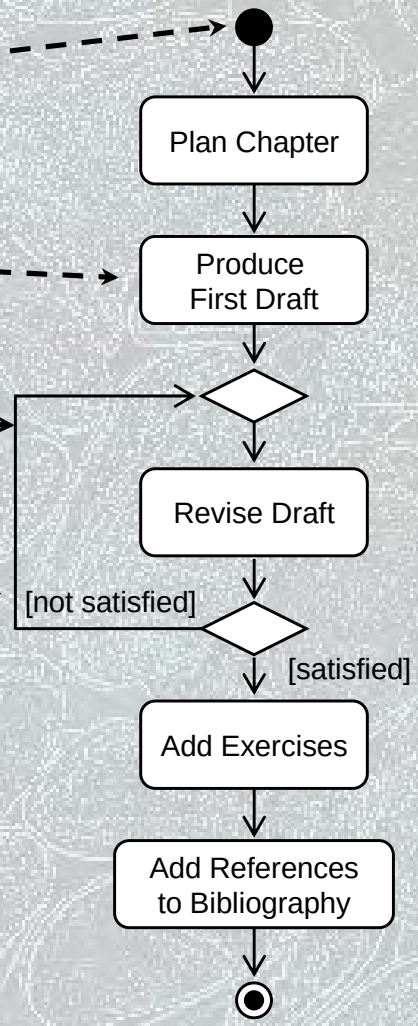


Hiding Detail



Diagrams in UML

- UML diagrams consist of:
  - icons
  - two-dimensional symbols
  - paths
  - Strings
- UML diagrams are defined in the UML specification.



Diagrams vs Models

- A diagram illustrates some aspect of a system.
- A model provides a complete view of a system at a particular stage and from a particular perspective.
- A model may consist of a single diagram, but most consist of many related diagrams and supporting data and documentation.

Examples of Models

- Requirements Model
  - complete view of requirements
  - may include other models, such as a Use Case Model
  - includes textual description as well as sets of diagrams

Examples of Models

- Behavioural Model
  - shows how the system responds to events in the outside world and the passage of time
  - an initial model may just use Communication Diagrams
  - a later model will include Sequence Diagrams and State Machines

Models in UML

- A system is the overall thing that is being modelled
- A subsystem is a part of a system consisting of related elements
- A model is an abstraction of a system or subsystem from a particular perspective
- A model is complete and consistent at the chosen level of abstraction

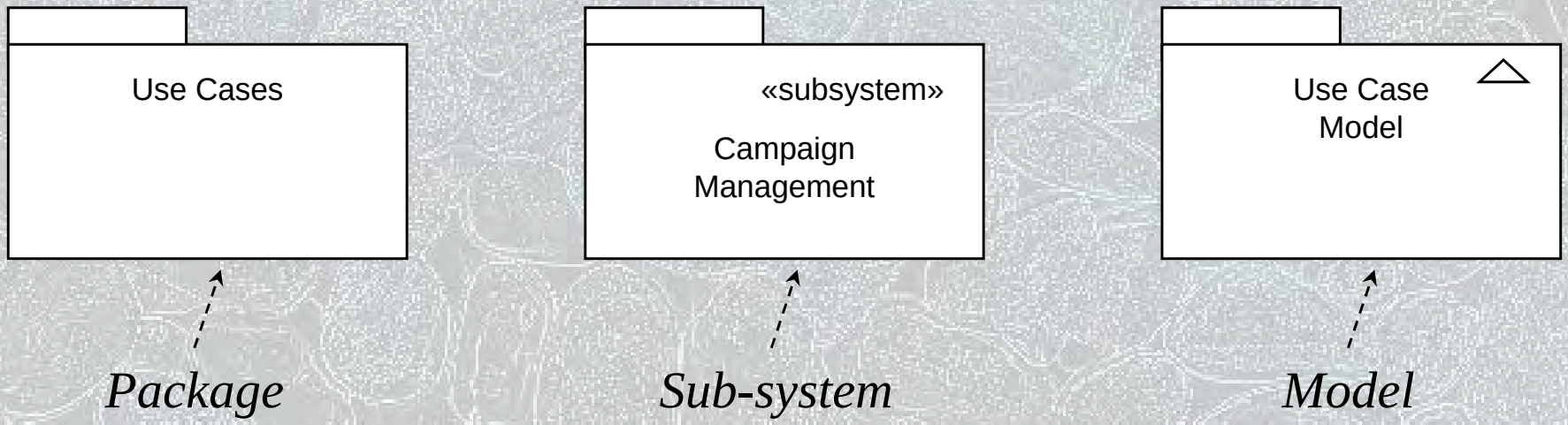
Models in UML

- Different models present different views of the system, for example:
  - use case view
  - design view
  - process view
  - implementation view
  - deployment view

(Booch et al., 1999)

Packages, Sub-systems and Models

- UML has notation for showing subsystems and models, and also for packages, which are a mechanism for organising models (e.g. in CASE tools)

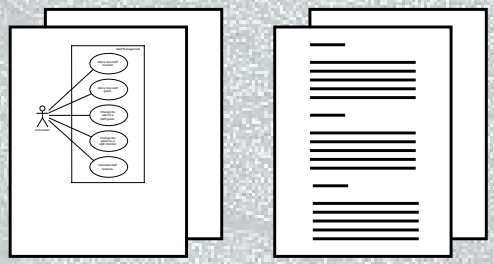


Developing Models

- During the life of a project using an iterative life cycle, models change along the dimensions of:
  - abstraction—they become more concrete
  - formality—they become more formally specified
  - level of detail—additional detail is added as understanding improves

Development of the Use Case Model

Iteration 1
Obvious use cases.
Simple use case descriptions.



Iteration 2
Additional use cases.
Simple use case descriptions.
Prototypes.

Iteration 3
Structured use cases.
Structured use case descriptions.
Prototypes.

Summary

In this lecture you have learned about:

- What is meant by a model
- The distinction between a model and a diagram
- The UML concept of a model

References

- Booch, Rumbaugh and Jacobson (1999)
- Bennett, Skelton and Lunn (2005)

(For full bibliographic details, see Bennett, McRobb and Farmer)

Activity Diagrams

Based on Chapter 5 of Bennett, McRobb and Farmer:

Object Oriented Systems Analysis and Design Using UML, (3rd Edition), McGraw Hill, 2005.

In This Lecture You Will Learn:

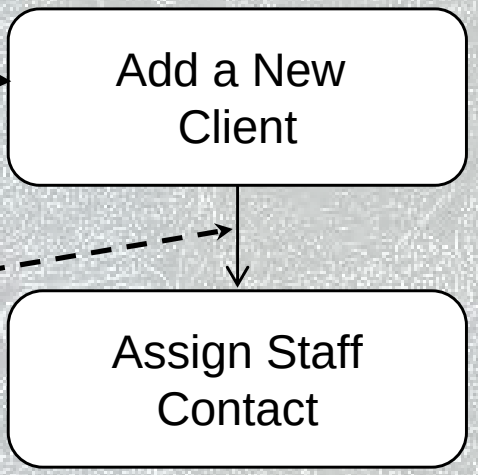
- The purpose of activity diagrams
- The notation of activity diagrams
- How to draw activity diagrams

Drawing Activity Diagrams

- Purpose
  - to model a task (for example in business modelling)
  - to describe a function of a system represented by a use case
  - to describe the logic of an operation
  - to model the activities that make up the life cycle in the Unified Process

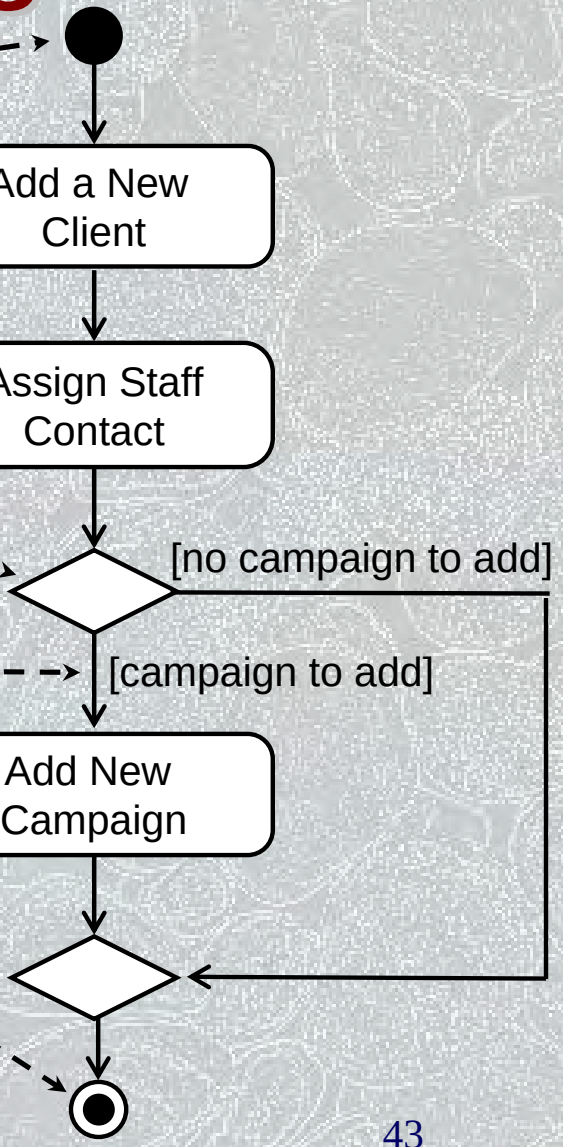
Notation of Activity Diagrams

- Actions
  - rectangle with rounded corners
  - meaningful name
- Control flows
  - arrows with open arrowheads



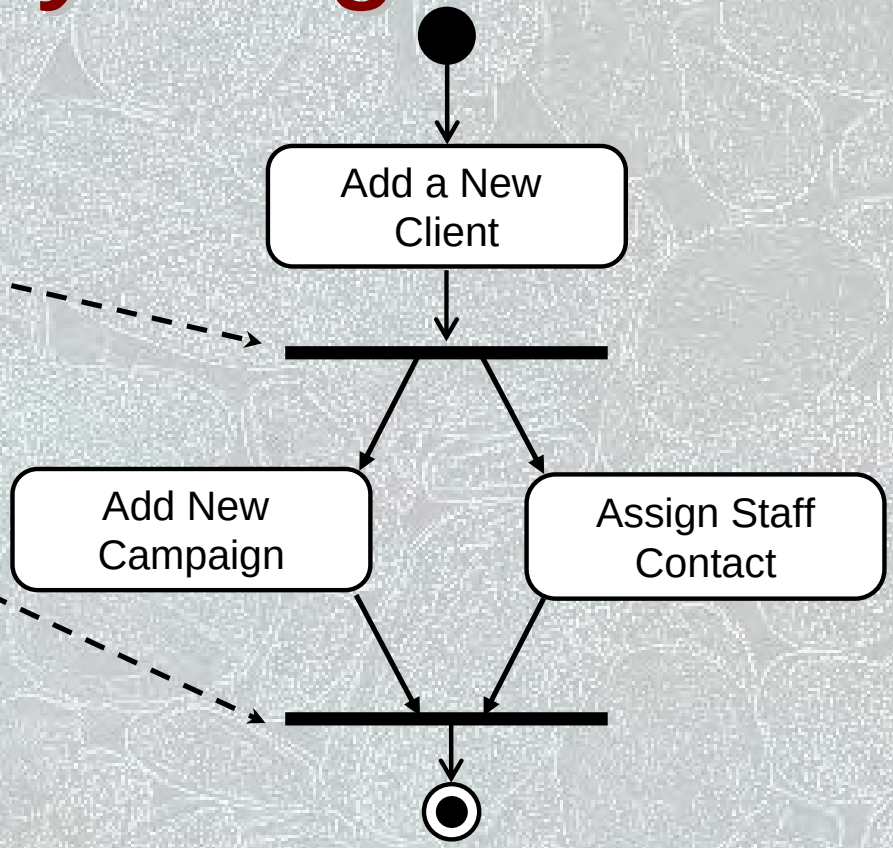
Notation of Activity Diagrams

- Initial node
  - black circle
- Decision nodes (and merge nodes)
  - diamond
- Guard conditions
  - in square brackets
- Final node
  - black circle in white circle



Notation of Activity Diagrams

- Fork nodes and join nodes
  - thick bar
- Actions carried out in parallel

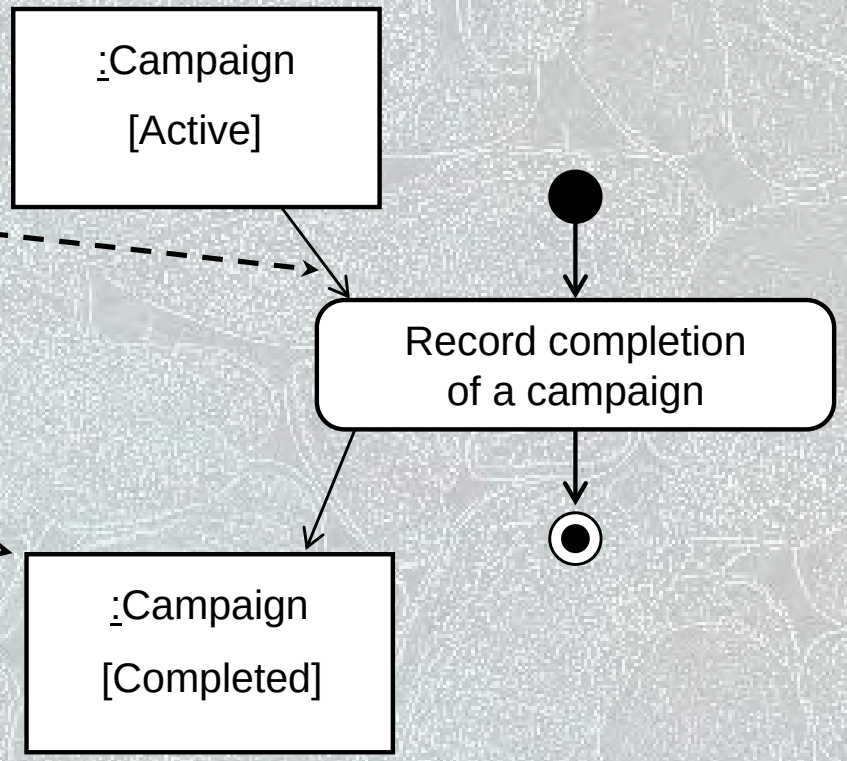


Notation of Activity Diagrams

- In UML 1.X multiple flows from an action were implicitly ORed
- In UML 2.0 they are implicitly ANDed
- Guard conditions do not have to be mutually exclusive, but it is advisable that they should be
- Decisions should be strictly nested, but...
- ... a merge point can be combined with a following decision point

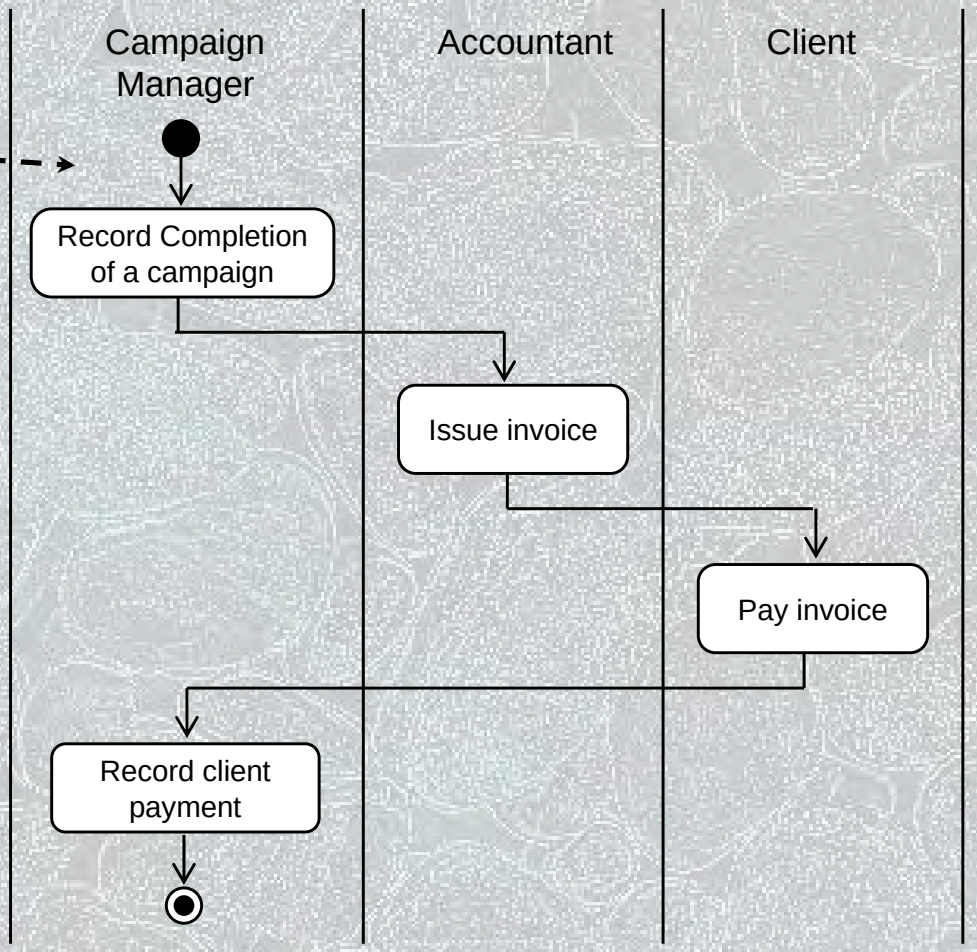
Notation of Activity Diagrams

- Object flows
  - open arrow
- Objects
  - rectangle
  - optionally shows the state of the object in square brackets



Notation of Activity Diagrams

- Activity Partitions (Swimlanes)
  - vertical columns
  - labelled with the person, organisation, department or system responsible for the activities in that column



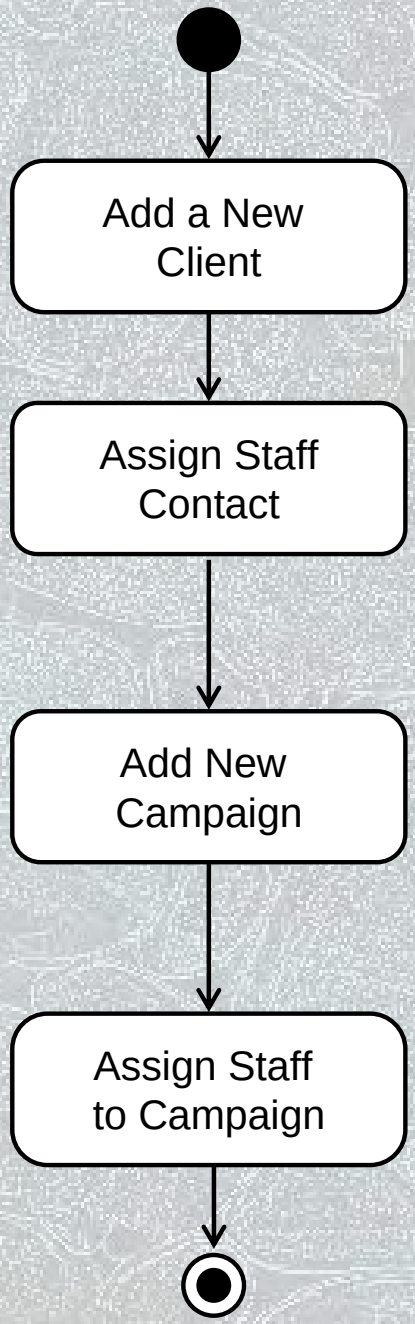
Drawing Activity Diagrams

- What is the purpose?
  - This will influence the kind of activities that are shown
- What is being shown in the diagram?
  - What is the name of the business process, use case or operation?
- What level of detail is required?
  - Is it high level or more detailed?

Drawing Activity Diagrams

- Identify actions
  - What happens when a new client is added in the Agate system?
    - Add a New Client
    - Assign Staff Contact
    - Add New Campaign
    - Assign Staff to Campaign
- Organise the actions in order with flows

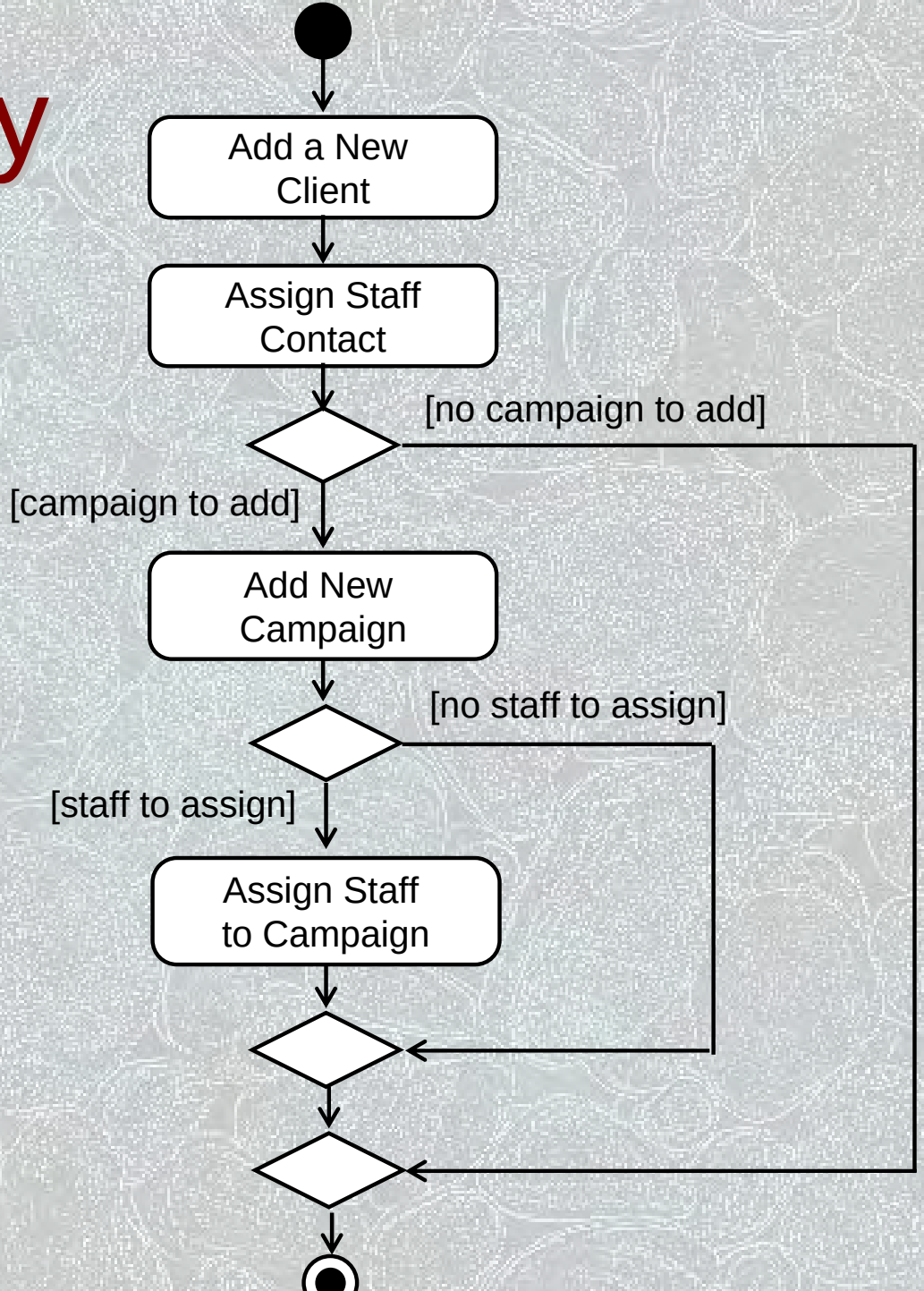
Drawing Activity Diagrams



Drawing Activity Diagrams

- Identify any alternative flows and the conditions on them
  - sometimes there is a new campaign to add for a new client, sometimes not
  - sometimes they will want to assign staff to the campaign, sometimes not
- Add decision and merge nodes, flows and guard conditions to the diagram

Drawing Activity Diagrams



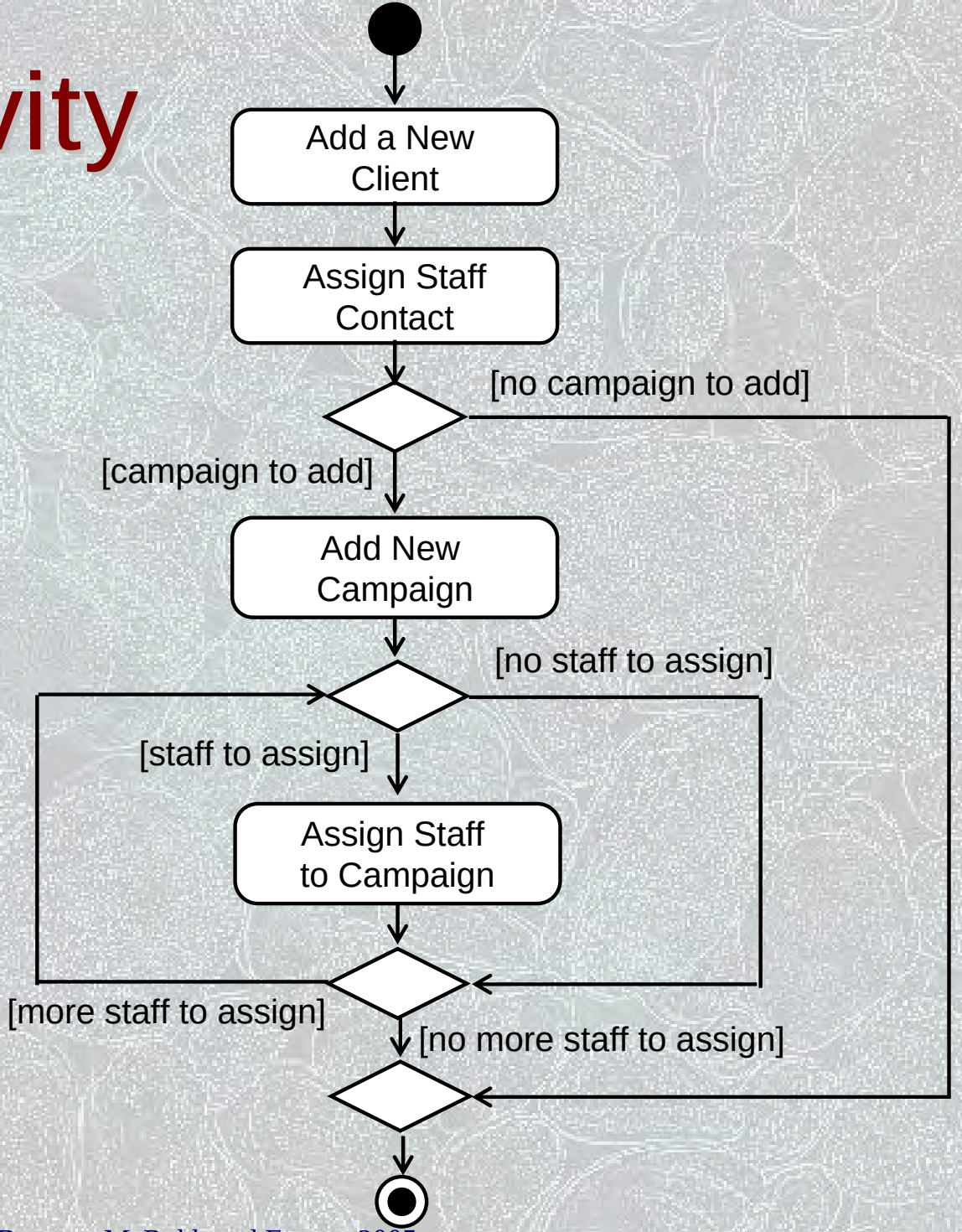
Drawing Activity Diagrams

- Identify any actions that are carried out in parallel
  - there are none in this example
- Add fork and join nodes and flows to the diagram

Drawing Activity Diagrams

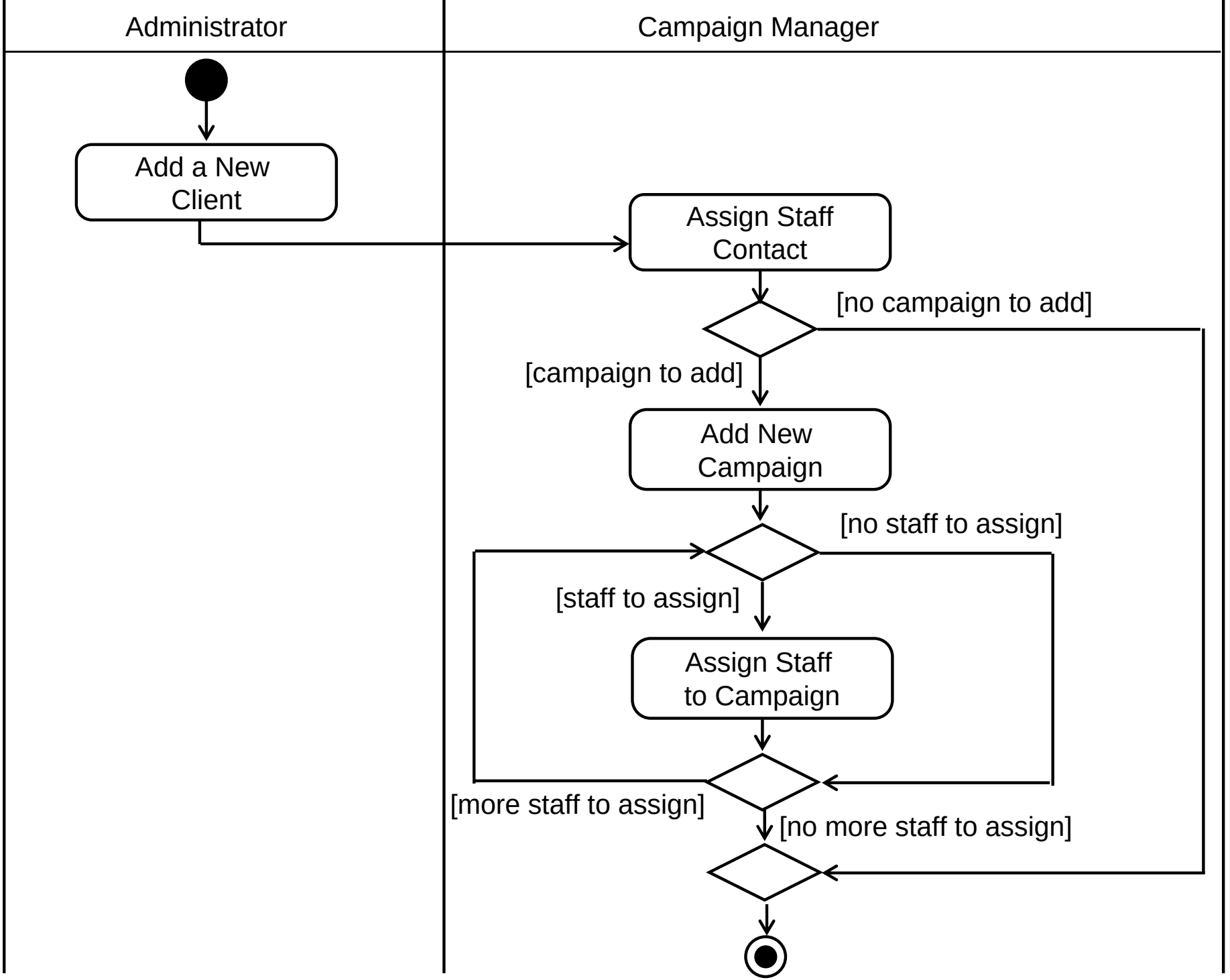
- Identify any processes that are repeated
  - they will want to assign staff to the campaign until there are no more staff to add
- Add decision and merge nodes, flows and guard conditions to the diagram

Drawing Activity Diagrams



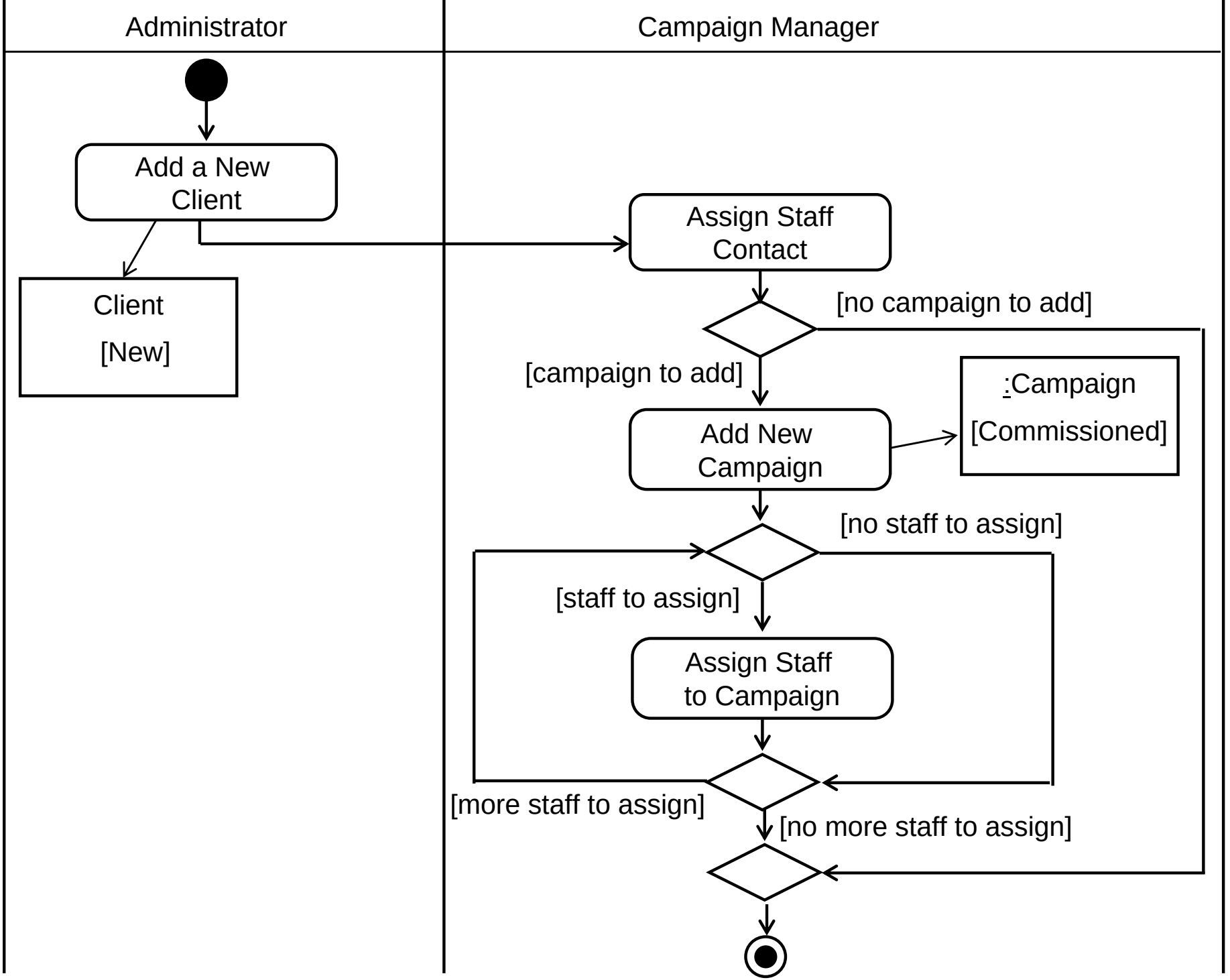
Drawing Activity Diagrams

- Are all the activities carried out by the same person, organisation or department?
- If not, then add swimlanes to show the responsibilities
- Name the swimlanes
- Show each activity in the appropriate swimlane


Administrator
Campaign Manager
Add a New Client
Assign Staff Contact
[no campaign to add]
[campaign to add]
Add New Campaign
[no staff to assign]
[staff to assign]
Assign Staff to Campaign
[more staff to assign]
[no more staff to assign]

Drawing Activity Diagrams

- Are there any object flows and objects to show?
  - these can be documents that are created or updated in a business activity diagram
  - these can be object instances that change state in an operation or a use case
- Add the object flows and objects


Administrator
Campaign Manager
Add a New Client
Client
[New]
Assign Staff Contact
[no campaign to add]
[campaign to add]
Add New Campaign
:Campaign
[Commissioned]
[no staff to assign]
[staff to assign]
Assign Staff to Campaign
[more staff to assign]
[no more staff to assign]

Summary

In this lecture you have learned about:

- The purpose of activity diagrams
- The notation of activity diagrams
- How to draw activity diagrams

References

- The notation and semantics of activity diagrams have changed significantly since UML was first released. The original UML books are now out of date on the subject.
- Bennett, Skelton and Lunn (2005)

(For full bibliographic details, see Bennett, McRobb and Farmer)

Development Process

Based on Chapter 5 of Bennett, McRobb and Farmer:

Object Oriented Systems Analysis and Design Using UML, (3rd Edition), McGraw Hill, 2005.

In This Lecture You Will Learn:

- About the Unified Software Development Process
- How phases relate to workflows in an iterative life cycle
- An approach to system development
- Major activities in the development process

Unified Software Development Process

- Developed by the team that created UML
- Embodies best practice in system development
- Adopts an iterative approach with four main phases
- Different tasks are captured in a series of workflows

Best Practice

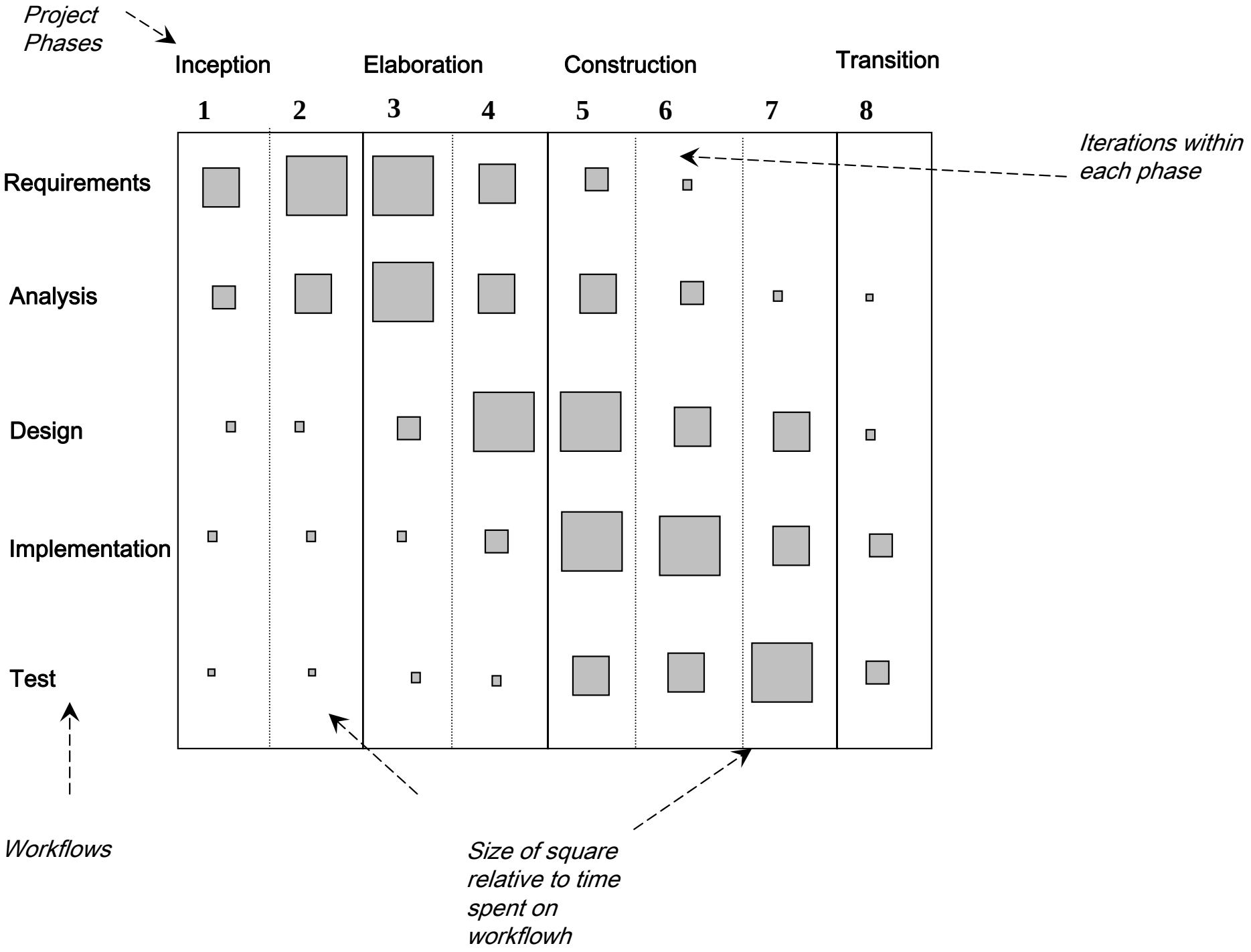
- Iterative and incremental development
- Component-based development
- Requirements-driven development
- Configurability
- Architecture-centrism
- Visual modelling techniques

Four Phases

- Inception
- Elaboration
- Construction
- Transition

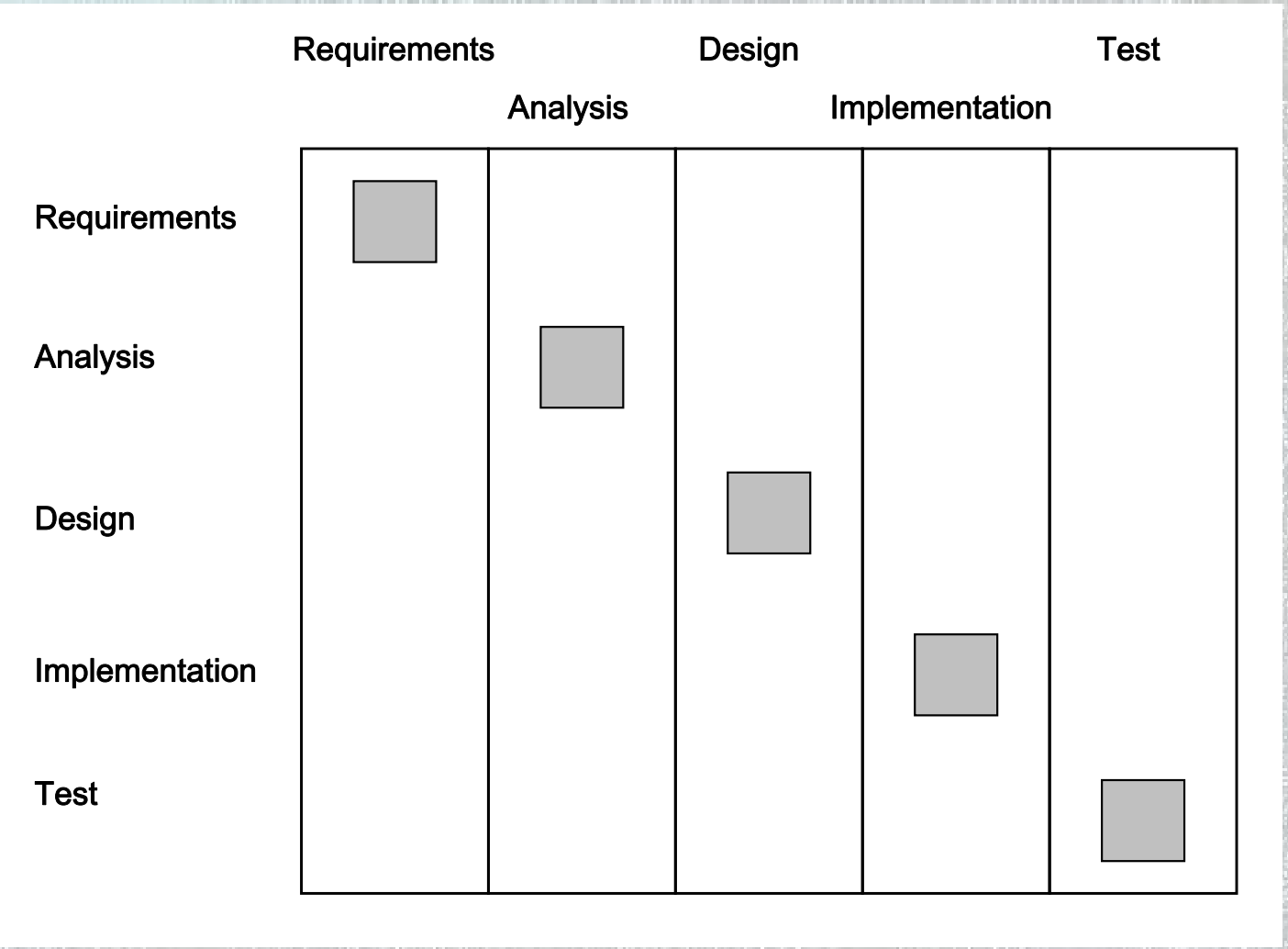
Phases, Workflows and Iterations

- Within each phase activities are grouped into workflows
- The balance of effort spent in each workflow varies from phase to phase
- Within phases there may be more than one iteration


Project Phases
Inception
Elaboration
Construction
Transition
1
2
3
4
5
6
7
8
Requirements
Analysis
Design
Implementation
Test
Iterations within each phase
Workflows
Size of square relative to time spent on workflowh

Difference from Waterfall Life Cycle

- In a waterfall life cycle project the phases and the workflows are linked together
- In the Requirements phase, only Requirements workflow activities are carried out
- All Requirements activity should be completed before work starts on Analysis
- In an iterative life cycle project it is recognised that some Requirements work will be happening alongside Analysis work



Major Activities of the Development Process

Activity	Techniques	Key Deliverables
Requirements Capture and Modelling	Requirements Elicitation Use Case Modelling Architectural Modelling Prototyping	Use Case Model Requirements List Initial Architecture Prototypes Glossary

Major Activities of the Development Process

Activity	Techniques	Key Deliverables
Requirements Analysis	Communication Diagrams Class and Object Modelling Analysis Modelling	Analysis Models

Major Activities of the Development Process

Activity	Techniques	Key Deliverables
System Design	Deployment Modelling Component Modelling Package Modelling Architectural Modelling Design Patterns	Overview Design and Implementation Architecture

Major Activities of the Development Process

Activity	Techniques	Key Deliverables
Class Design	Class and Object Modelling Interaction Modelling State Modelling Design Patterns	Design Models

Major Activities of the Development Process

Activity	Techniques	Key Deliverables
User Interface Design	Class and Object Modelling Interaction Modelling State Modelling Package Modelling Prototyping Design Patterns	Design Models with Interface Specification

Major Activities of the Development Process

Activity	Techniques	Key Deliverables
Data Management Design	Class and Object Modelling Interaction Modelling State Modelling Package Modelling Design Patterns	Design Models with Database Specification

Major Activities of the Development Process

Activity	Techniques	Key Deliverables
Construction	Programming Component Re-use Database DDL Programming Idioms Manual Writing	Constructed System Documentation

Major Activities of the Development Process

Activity	Techniques	Key Deliverables
Testing	Programming Test Planning and Design Testing	Test Plans Test Cases Tested System

Major Activities of the Development Process

Activity	Techniques	Key Deliverables
Implementation	Planning Training Data Conversion	Installed System

Summary

In this lecture you have learned about:

- The Unified Software Development Process
- How phases relate to workflows in an iterative life cycle
- An approach to system development
- Major activities in the development process

References

- Jacobson, Booch and Rumbaugh (1999)
- Kruchten (2004)
- Chapter 21 of Bennett, McRobb and Farmer includes more detail about the Unified Process

(For full bibliographic details, see Bennett, McRobb and Farmer)